Dylan Wulf

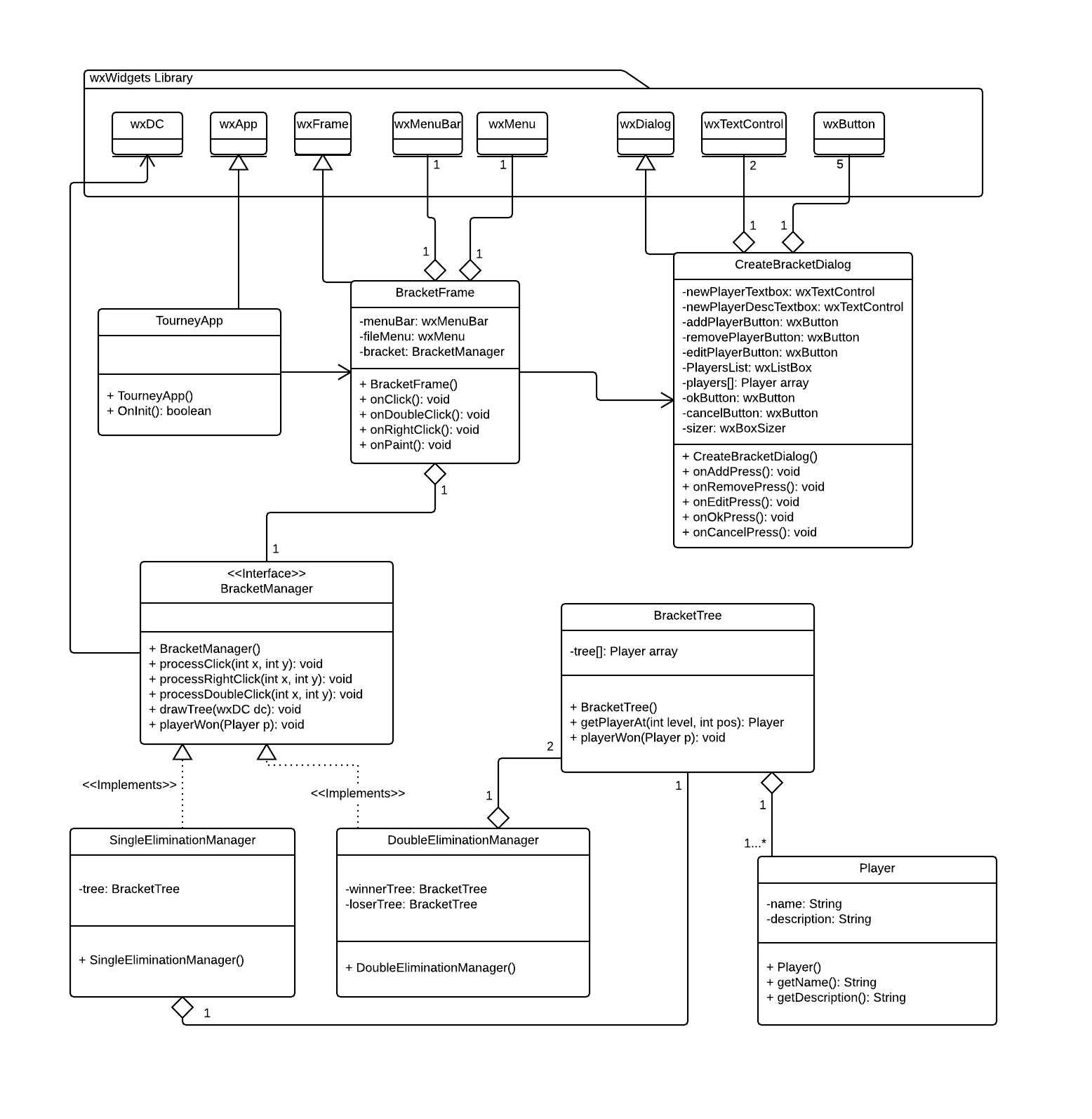
CSC 415

March 31, 2015

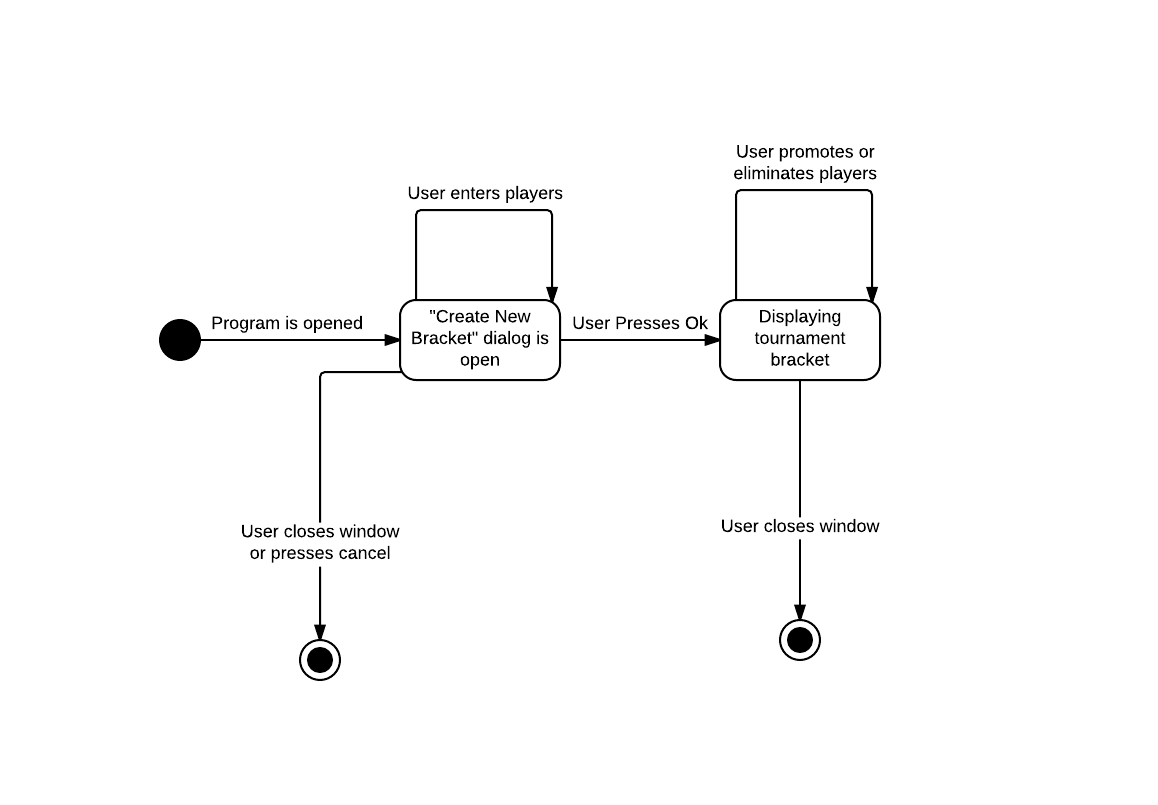
Assignment 4 – Open Source Software: Analysis and Design

Project Design

**Design Class Diagram**

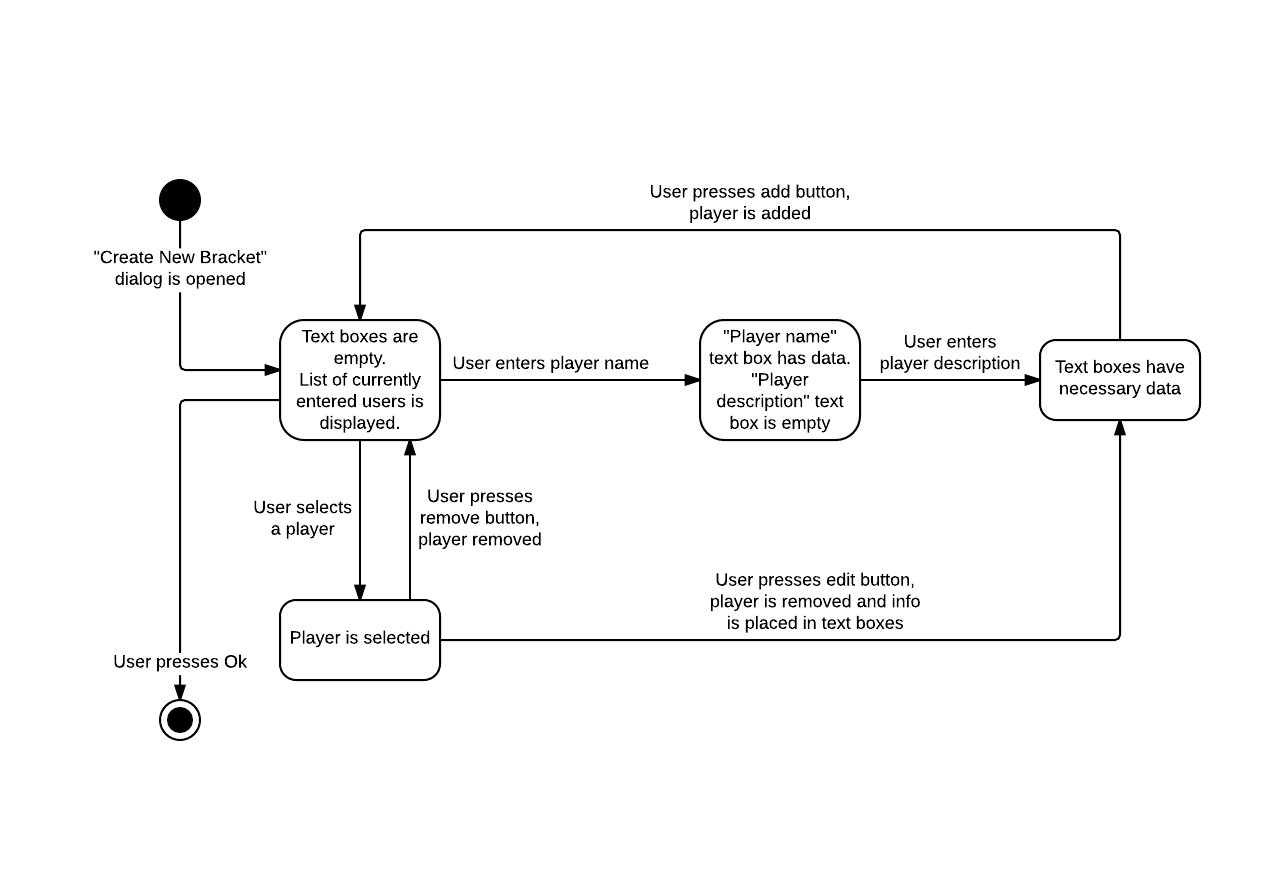


**Overall Statechart**

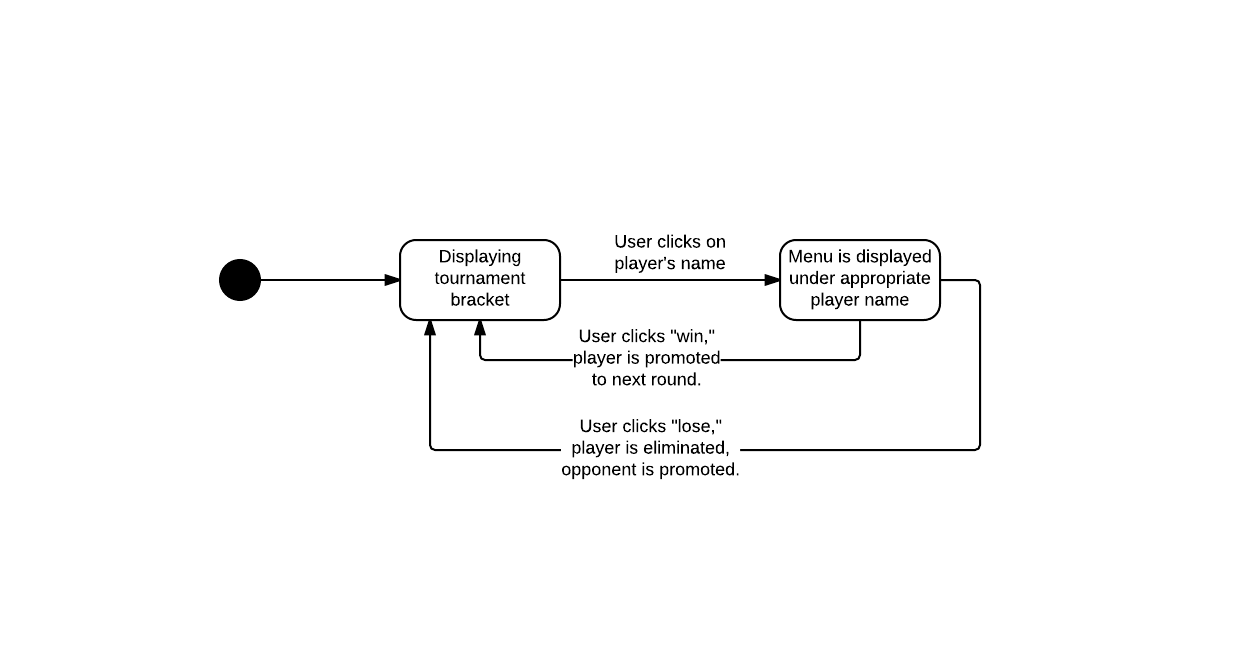


**Specific Statecharts**

Entering new players

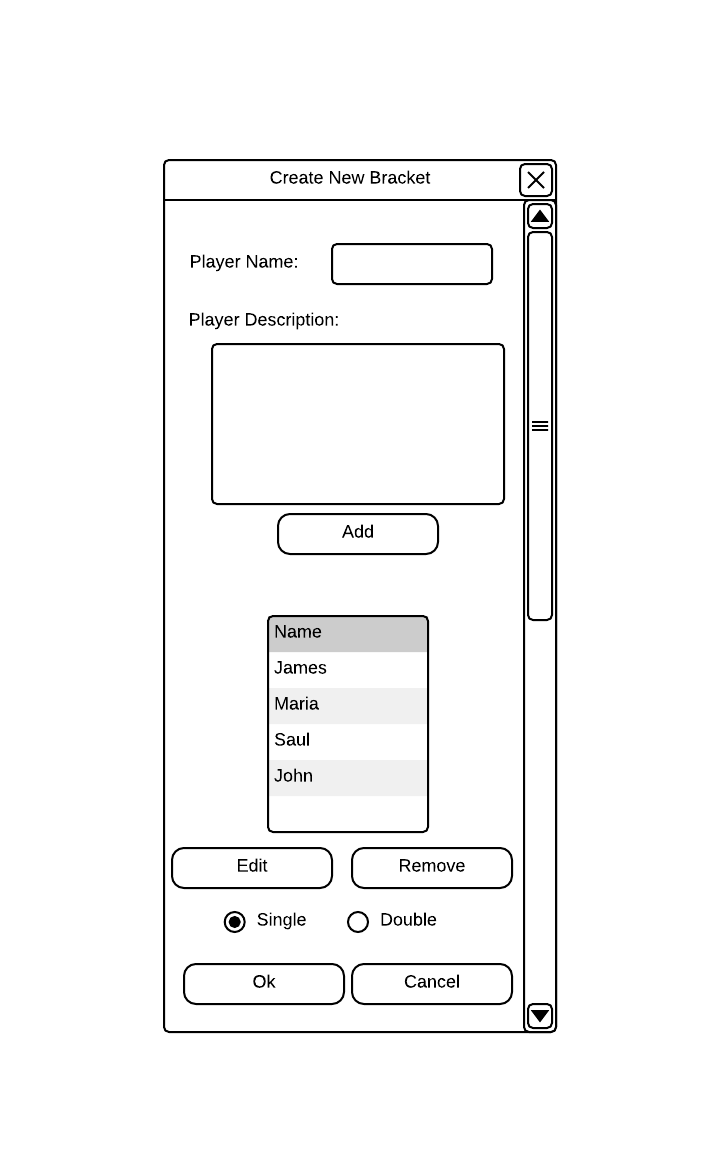
****

Promoting/eliminating players

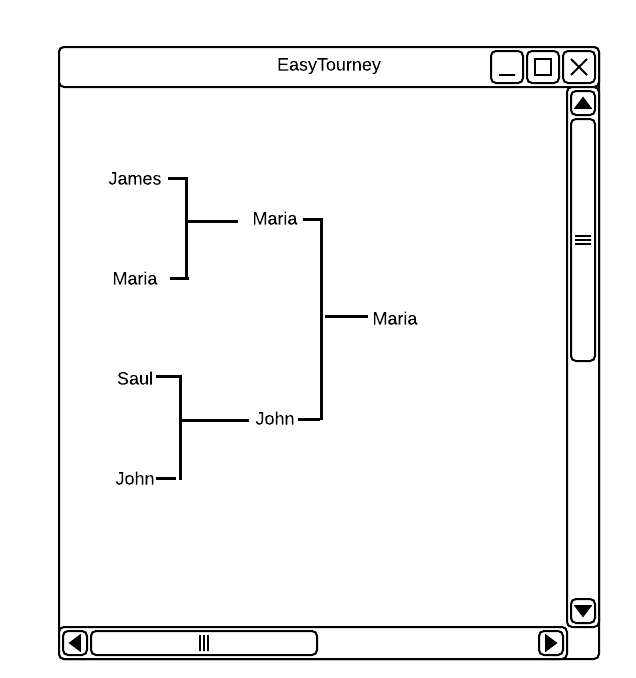
****

**User Interface**

Create New Bracket Dialog



Main Window where bracket is displayed



The User Interface will abide by the “Eight golden rules” in the following ways:

* Consistency—The program will maintain the same look and feel in all windows, and will keep the native look and feel of the OS platform
* Shortcuts—Standard keyboard shortcuts will be available anywhere there is text
* Informative feedback—When the user clicks on a button, it will change appearance.
* Simple Error Handling—When something goes wrong, a simple popup box will show with the error
* Easy Reversal of Actions—When entering players, the user can remove/edit players already entered.
* Internal Locus of control—Window will be resizable to whatever size the user wants
* Reduce short-term memory load—All of the relevant information is displayed on the screen, so there is no need to memorize anything

Test Case Design

A lot of my program’s functionality depends on GUI elements, which cannot be easily tested with unit and integration testing. In order to test these graphical elements, the program (or at least part of it) must be able to compile and run in order to visually confirm that they are operating correctly. I have not found many easy-to-use GUI testing tools, and since my program does not have a very complicated GUI, I can do the user interface testing manually.

During unit testing, I will test all of the individual classes that do not depend on GUI elements, such as Player and BracketTree. The BracketTree class is what manages the binary tree structure, so this class will probably receive the most unit testing. Then once I confirm that all the classes work separately, I can start integration testing by piecing them together one by one and testing them together.

Tools: Google Test for testing and Valgrind for debugging

Test Cases

|  |  |  |  |
| --- | --- | --- | --- |
| **Functionality Tested** | **Inputs** | **Expected Output** | **Actual Output** |
| Adding player | Player name in text box, player description in text box, press “add” button | Text boxes will clear and player name will appear in list |  |
| Editing player | Player selected, press “edit” button | Player disappears from list, player info appears in text boxes |  |
| Removing player | Player selected, press “remove” button | Player disappears from list |  |
| List scrolling | When multiple players are added, the list can be scrolled so that all players can be viewed |  |  |
| Cancel button | Cancel button is pressed | Program will close |  |
| Initial bracket creation | OK button is pressed after players have been entered | Bracket creation dialog will close, main bracket window will become focused, all players are in the bracket, there are no duplicates, the whole bracket is viewable, everything is in the right place |  |
| Error handling: no players | OK button is pressed when no players have been entered | Popup error message |  |
| Player menu showing up | Player name is clicked on in main bracket window | Menu appears with win/lose options |  |
| Player menu functionality | Win or lose option is selected | Performs the correct operation to the correct player and updates the bracket accordingly |  |
| BracketTree class: getPlayerAt | getPlayerAt(x, y) is called with any valid position in the tree. | getPlayerAt(0,0) returns Player in bottom left of tree,  any other valid position returns the correct Player in that spot in the tree |  |
| BracketTree class: playerWon | playerWon(p) is called with any player currently in the tree | The Player p was correctly promoted to the next level in the tree |  |

Github:

ID: wulfd1

Repository: EasyTourney